LING 364: Introduction to Formal Semantics

Lecture 11

February 16th

Administrivia

- Today: Special Computer Lab Class
  - Homework 2 help session
- Reminder
  - Extension: Homework 2 due tonight
  - strict deadline

Tools we need

- *from earlier lecture slides...*

Adding Phrase Structure

- Modify basic DCG into one that includes phrase structure

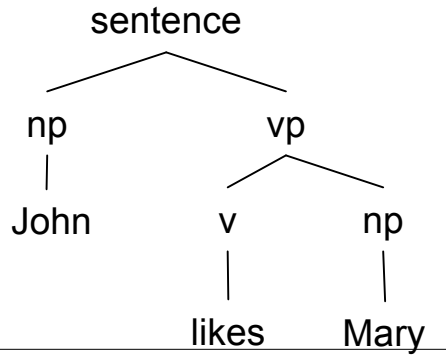
- **Basic DCG:**

```
sentence --> np, vp.
vp --> v, np.
v --> [likes].
np --> [john].
np --> [mary].
```

- **Query**: (we supply two arguments: sentence as a list and an empty list)

```
?- sentence([john,likes,mary],[]).
Yes (Answer)
```

```
sentence(np(john),vp(v(likes),np(mary)))
```



- **Phrase Structure DCG:**

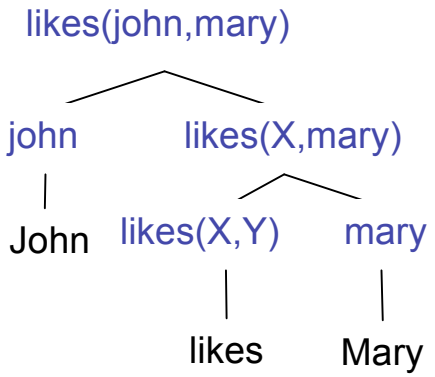
```
sentence(sentence(NP,VP)) --> np(NP), vp(VP).
vp(vp(V,NP)) --> v(V), np(NP).
v(v(likes)) --> [likes].
np(np(john)) --> [john].
np(np(mary)) --> [mary].
```

- **Modified Query**: (supply one more argument)

```
?- sentence(PS,[john,likes,mary],[]).
PS = sentence(np(john),vp(v(likes),np(mary)))
```

Adding Meaning

- modify basic DCG into one that includes meaning



- **Basic DCG:**

```
sentence --> np, vp.
vp --> v, np.
v --> [likes].
np --> [john].
np --> [mary].
```

- **Query**: (we supply two arguments: sentence as a list and an empty list)

```
?- sentence([john,likes,mary],[]).
Yes (Answer)
```

argument saturation

`arg(Nth,Predicate,Argument)` means make `Nth` argument of `Predicate` equal to `Argument`

{ `<Goal>` } means call Prolog `<Goal>`
`{arg(2,VBm,NPm)}` means call `arg(2,VBm,NPm)`

- **Meaning DCG:**
  - `sentence(P) --> np(NP1), vp(P), {saturate1(P,NP1)}.`
  - `vp(P) --> v(P), np(NP2), {saturate2(P,NP2)}.`
  - `v(likes(X,Y)) --> [likes].`
  - `np(john) --> [john].`
  - `np(mary) --> [mary].`
  - `saturate1(P,A) :- arg(1,P,A).`
  - `saturate2(P,A) :- arg(2,P,A).`
- **Query**: (supply one more argument)
- `?- sentence(M,[john,likes,mary],[]).`
  `M = likes(john,mary)`

Recursion

- **Order of grammar rules matters in Prolog...**
- **(Fixed) Prolog Computation Rule:**
  - always pick the *first-mentioned* matching grammar rule to try each time we expand a non-terminal
- **General Rule for writing recursive rules:**
  - put recursive case **last**
  - i.e. *place non-recursive rules for a non-terminal ahead of the recursive ones*
  - avoid Infinite Loop in Prolog
  - ERROR: out of local stack.

Homework 2

- Exercises 1 through 3

- we built a basic DCG grammar for this in the extra lab session last Thursday

Exercises 1 through 3

- Give a **basic** DCG grammar that covers the following sentences and questions
  - $[_{Sbar}[_{S}\ [_{NP}$ John$]\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{DET}$ a$][_{N}$ student$]]]]]$
  - $[_{Sbar}[_{S}\ [_{NP}$ Pete$]\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{DET}$ a$][_{N}$ student$]]]]]$
  - $[_{Sbar}[_{S}\ [_{NP}$ Mary$]\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{DET}$ a$][_{N}$ baseball fan$]]]]]$
  - $[_{Sbar}[_{S}\ [_{NP}$ Pete$]\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{DET}$ a$][_{N}$ baseball fan$]]]]]$
  - $[_{Sbar}[_{S}\ [_{NP}$ John$]\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{DET}$ a$][_{N}$ baseball fan$]]]]]$
  - $[_{Sbar}\ [_{NP}$ Who$]\ [_{S}\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{DET}$ a$][_{N}$ student$]]]]]$
  - $[_{Sbar}\ [_{NP}$ Who$]\ [_{S}\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{DET}$ a$][_{N}$ baseball fan$]]]]]$
  - $[_{Sbar}\ [_{NP}$ Who$]\ [_{S}\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{NEG}$ not$]\ [_{NP}\ [_{DET}$ a$][_{N}$ student$]]]]]]$
  - $[_{Sbar}\ [_{NP}$ Who$]\ [_{S}\ [_{VP}\ [_{V}$ is$][_{NP}\ [_{NEG}$ not$]\ [_{NP}\ [_{DET}$ a$][_{N}$ baseball fan$]]]]]]$
  - $[_{Sbar}\ [_{NP}$ Who$]\ [_{S}\ [_{VP}\ [_{V}$ is$]\ [_{NP}[_{NP}\ [_{DET}$ a$][_{N}$ student$]]]]][_{CONJ}$ and$][_{NP}\ [_{DET}$ a$][_{N}$ baseball fan$]]]]]]$
  - $[_{Sbar}\ [_{NP}$ Who$]\ [_{S}\ [_{VP}\ [_{V}$ is$]\ [_{NP}[_{NP}\ [_{DET}$ a$][_{N}$ student$]]]]][_{CONJ}$ and$][_{NP}\ [_{NEG}$ not$][_{NP}[_{DET}$ a$][_{N}$ baseball fan$]]]]]]]$

Sample Grammar

```
% Exercises 1 through 3
sbar --> np, s.
sbar --> s.
s --> vp.
s --> np, vp.
np --> [john].
np --> [pete].
np --> [mary].
np --> det, n.
np --> [who].
```

- `np --> neg, np.`
- `np --> np, conj, np.`
- `n --> [student].`
- `n --> [baseball,fan].`
- `neg --> [not].`
- `conj --> [and].`
- `vp --> v, np.`
- `v --> [is].`
- `det --> [a].`

Exercise 4

- Modify the grammar obtained so far, i.e. by Exercise 3, to include phrase structure
- Show your grammar produces phrase structure for the previously mentioned sentences and questions

```
| ?- sbar(PS,[who,is,not,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(neg(not),np(det(a),n(baseball_fan))))))) ?
| ?- sbar(PS,[john,is,a,baseball,fan],[]).
PS = sbar(s(np(john),vp(v(is),np(det(a),n(baseball_fan))))))) ?
| ?- sbar(PS,[who,is,a,student,and,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(np(det(a),n(student)),conj(and),np(det(a),
n(baseball_fan))))))) ?
| ?- sbar(PS,[who,is,a,student,and,not,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(np(det(a),n(student)),conj(and),np(neg(not),
np(det(a),n(baseball_fan)))))))) ?
```

Exercise 4

- Step 1: Get some phrase structure output first
- Step 2: Note:
  - rule ordering is important here...
  - how to ensure the right rule gets applied for sentences vs. questions?
  - `sbar --> s.`
  - `sbar --> np, s.`

```
| ?- sbar(PS,[who,is,not,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(neg(not),np(det(a),n(baseball_fan))))))) ?
| ?- sbar(PS,[john,is,a,baseball,fan],[]).
PS = sbar(s(np(john),vp(v(is),np(det(a),n(baseball_fan))))))) ?
| ?- sbar(PS,[who,is,a,student,and,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(np(det(a),n(student)),conj(and),np(det(a),
n(baseball_fan))))))) ?
| ?- sbar(PS,[who,is,a,student,and,not,a,baseball,fan],[]).
PS = sbar(np(who),s(vp(v(is),np(np(det(a),n(student)),conj(and),np(neg(not),
np(det(a),n(baseball_fan)))))))) ?
```

Exercise 5

- Modify the grammar obtained so far, i.e. by Exercise 3, to generate meaning
  - e.g.
    - `student(mary).`
    - `student(X), \+ baseball_fan(X).`
- Show your grammar produces appropriate meanings for the previously mentioned sentences and questions

```
| ?- sbar(M,[who,is,not,a,baseball,fan],[]).
M = \+baseball_fan(_A) ?
| ?- sbar(M,[john,is,a,baseball,fan],[]).
M = baseball_fan(john) ?
| ?- sbar(M,[who,is,a,student,and,a,baseball,fan],[]).
M = student(_A),baseball_fan(_A) ?
| ?- sbar(M,[who,is,a,student,and,not,a,baseball,fan],[]).
M = student(_A),\+baseball_fan(_A) ?
```

Note:
_A is an internally-generated Prolog variable

Exercise 5

- Step 1: Generate some meaning output.
- Step 2: Notice, for example, that for the conjunction cases you need the same variable for both predicate nominals
  - i.e. `saturate1/2` must be modified to deal with logical connectives like `,` (conjunction) and `\+` (negation)

```
| ?- sbar(M,[who,is,not,a,baseball,fan],[]).
M = \+baseball_fan(_A) ?
| ?- sbar(M,[john,is,a,baseball,fan],[]).
M = baseball_fan(john) ?
| ?- sbar(M,[who,is,a,student,and,a,baseball,fan],[]).
M = student(_A),baseball_fan(_A) ?
| ?- sbar(M,[who,is,a,student,and,not,a,baseball,fan],[]).
M = student(_A),\+baseball_fan(_A) ?
```